In [1]:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
sns.set_theme(color_codes=True)
```

In [2]:

```
df = pd.read_csv(r'D:/project/Car Price Prediction/Cars Price Prediction.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	Car Make	Car Model	Year	Engine Size (L)	Horsepower	Torque (lb-ft)	0-60 MPH Time (seconds)	Price (in USD)
0	Porsche	911	2022	3	379	331	4	101,200
1	Lamborghini	Huracan	2021	5.2	630	443	2.8	274,390
2	Ferrari	488 GTB	2022	3.9	661	561	3	333,750
3	Audi	R8	2022	5.2	562	406	3.2	142,700
4	McLaren	720S	2021	4	710	568	2.7	298,000

In [4]:

```
df.select_dtypes(include='object').nunique()
```

Out[4]:

```
Car Make                    38
Car Model                  176
Engine Size (L)             45
Horsepower                 124
Torque (lb-ft)              93
0-60 MPH Time (seconds)     43
Price (in USD)             367
dtype: int64
```

In [5]:

```
# remove commas from the Price column
df['Price (in USD)'] = df['Price (in USD)'].str.replace(',', '')
# convert the Price column to integer
df['Price (in USD)'] = df['Price (in USD)'].astype(int)
df['Price (in USD)'].dtypes
```

Out[5]:

```
dtype('int32')
```

In [6]:

```
df.head()
```

Out[6]:

	Car Make	Car Model	Year	Engine Size (L)	Horsepower	Torque (lb-ft)	0-60 MPH Time (seconds)	Price (in USD)
0	Porsche	911	2022	3	379	331	4	101200
1	Lamborghini	Huracan	2021	5.2	630	443	2.8	274390
2	Ferrari	488 GTB	2022	3.9	661	561	3	333750
3	Audi	R8	2022	5.2	562	406	3.2	142700
4	McLaren	720S	2021	4	710	568	2.7	298000

In [7]:

```
# remove commas from 0-60 MPH Time (seconds) column
df['0-60 MPH Time (seconds)'] = df['0-60 MPH Time (seconds)'].str.replace(',', '')

# check if the Horsepower column contains string values
if df['Horsepower'].dtype == 'object':
    # remove strings from the Column A column
    df['Horsepower'] = df['Horsepower'].str.replace('[^0-9]+', '', regex=True)

    # convert the remaining values to integers and fill empty strings with 0
    df['Horsepower'] = df['Horsepower'].apply(lambda x: int(x) if x != '' else 0)
else:
    # do something else if the Column A column does not contain string values
    pass

# check if the Horsepower column contains string values
if df['Torque (lb-ft)'].dtype == 'object':
    # remove strings from the Column A column
    df['Torque (lb-ft)'] = df['Torque (lb-ft)'].str.replace('[^0-9]+', '', regex=True)

    # convert the remaining values to integers and fill empty strings with 0
    df['Torque (lb-ft)'] = df['Torque (lb-ft)'].fillna(0)
    df['Torque (lb-ft)'] = df['Torque (lb-ft)'].apply(lambda x: int(x) if x != '' else 0
else:
    # do something else if the Column A column does not contain string values
    pass

# check if the Horsepower column contains string values
if df['0-60 MPH Time (seconds)'].dtype == 'object':
    # remove strings from the Column A column
    df['0-60 MPH Time (seconds)'] = df['0-60 MPH Time (seconds)'].str.replace('[^0-9]+',

    # convert the remaining values to integers and fill empty strings with 0
    df['0-60 MPH Time (seconds)'] = df['0-60 MPH Time (seconds)'].apply(lambda x: int(x)
else:
    # do something else if the Column A column does not contain string values
    pass


# Change numerical data into integer
df['Horsepower'] = df['Horsepower'].astype(int)
df['Torque (lb-ft)'] = df['Torque (lb-ft)'].astype(int)
df['0-60 MPH Time (seconds)'] = df['0-60 MPH Time (seconds)'].astype(float)

# convert the remaining values to integers and fill empty strings with 0
#df['Torque (lb-ft)'] = df['Torque (lb-ft)'].apply(lambda x: int(x) if x != '' else 0)
```

In [8]:

```
df.dtypes
```

Out[8]:

```
Car Make                    object
Car Model                   object
Year                         int64
Engine Size (L)             object
Horsepower                   int32
Torque (lb-ft)               int32
0-60 MPH Time (seconds)    float64
Price (in USD)               int32
dtype: object
```

In [9]:

```
df['Engine Size (L)'].unique()
```

Out[9]:

```
array(['3', '5.2', '3.9', '4', '4.4', '6.2', '3.8', '8', '5', '3.5',
       '4.7', '2', '2.9', '6', 'Electric', '6.5', '3.7', 'Electric Motor',
       '2.5', '1.5 + Electric', '6.8', '8.4', nan, '6.6', '7', '1.7',
       '3.3', '-', '6.7', '1.8', 'Electric (tri-motor)', '5.5',
       'Electric (93 kWh)', 'Electric (100 kWh)', 'Hybrid (4.0)', '4.6',
       '3.6', '1.5', 'Hybrid', '5.7', '2.0 (Electric)', '4.0 (Hybrid)',
       '0', '6.4', '6.3', '2.3'], dtype=object)
```

In [10]:

```
def segment_engine_size(engine_size):
    if engine_size in ['Electric', 'Hybrid']:
        return 'Electric/Hybrid'
    elif engine_size in ['Electric Motor', 'Electric (tri-motor)', 'Electric (93 kWh)',
        return 'Electric'
    elif engine_size == '1.5 + Electric':
        return '1.5 Hybrid'
    elif engine_size in ['Hybrid (4.0)', '4.0 (Hybrid)']:
        return '4.0 Hybrid'
    elif engine_size == '0':
        return 'Unknown'
    elif engine_size == '-':
        return 'Unknown'
    elif float(engine_size) < 2:
        return 'Small'
    elif float(engine_size) < 3:
        return 'Medium'
    else:
        return 'Large'
df['Engine Size (L)'] = df['Engine Size (L)'].apply(segment_engine_size)
```

In [11]:

```
df['Engine Size (L)'].unique()
```

Out[11]:

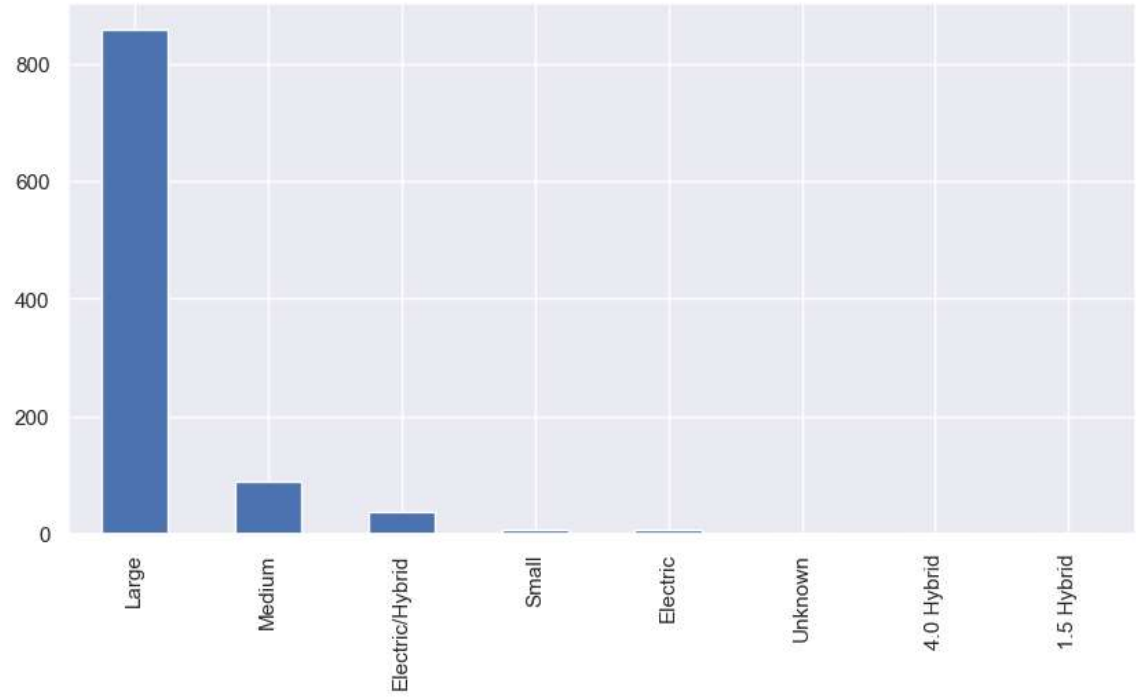
```
array(['Large', 'Medium', 'Electric/Hybrid', 'Electric', '1.5 Hybrid',
       'Small', 'Unknown', '4.0 Hybrid'], dtype=object)
```

In [12]:

```
plt.figure(figsize=(10,5))
df['Engine Size (L)'].value_counts().plot(kind='bar')
```

Out[12]:

```
<Axes: >
```



In [13]:

```
df.drop(columns='Car Model', inplace=True)
df.head()
```

Out[13]:

	Car Make	Year	Engine Size (L)	Horsepower	Torque (lb-ft)	0-60 MPH Time (seconds)	Price (in USD)
0	Porsche	2022	Large	379	331	4.0	101200
1	Lamborghini	2021	Large	630	443	28.0	274390
2	Ferrari	2022	Large	661	561	3.0	333750
3	Audi	2022	Large	562	406	32.0	142700
4	McLaren	2021	Large	710	568	27.0	298000

In [14]:

```
df['Car Make'].unique()
```

Out[14]:

```
array(['Porsche', 'Lamborghini', 'Ferrari', 'Audi', 'McLaren', 'BMW',
       'Mercedes-Benz', 'Chevrolet', 'Ford', 'Nissan', 'Aston Martin',
       'Bugatti', 'Dodge', 'Jaguar', 'Koenigsegg', 'Lexus', 'Lotus',
       'Maserati', 'Alfa Romeo', 'Ariel', 'Bentley', 'Mercedes-AMG',
       'Pagani', 'Polestar', 'Rimac', 'Acura', 'Mazda', 'Rolls-Royce',
       'Tesla', 'Toyota', 'W Motors', 'Shelby', 'TVR', 'Subaru',
       'Pininfarina', 'Kia', 'Alpine', 'Ultima'], dtype=object)
```

In [15]:

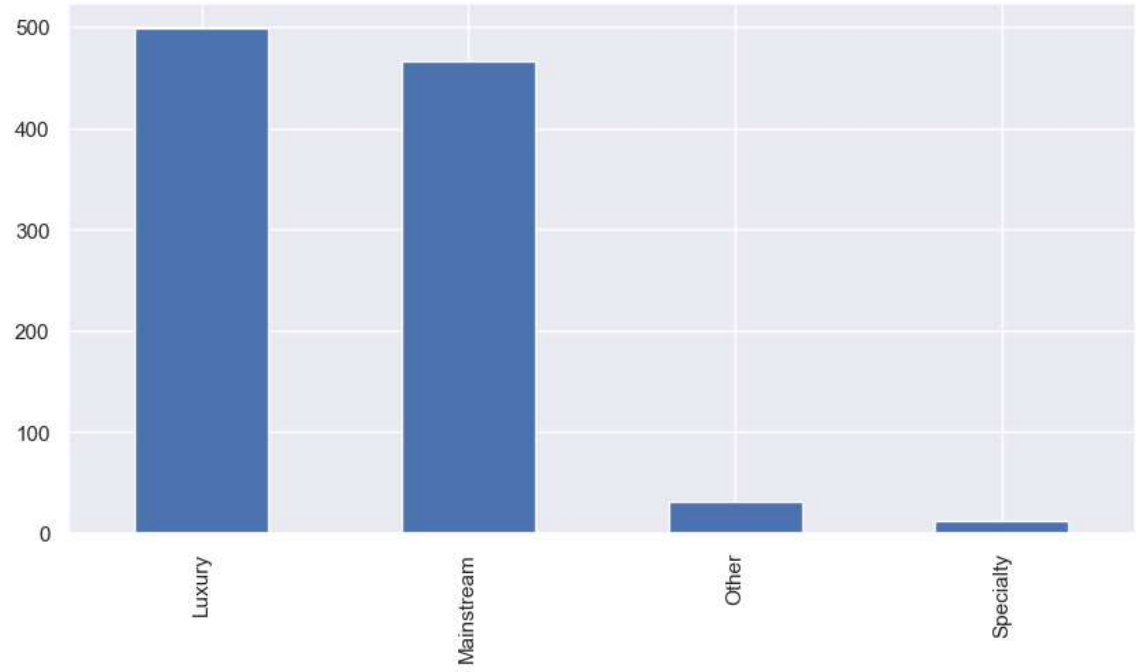
```
# define a function to segment the values
def segment_car_make(value):
    if value in ['Porsche', 'Lamborghini', 'Ferrari', 'McLaren', 'Aston Martin', 'Bugatt
        return 'Luxury'
    elif value in ['Audi', 'BMW', 'Mercedes-Benz', 'Chevrolet', 'Ford', 'Nissan', 'Dodge
        return 'Mainstream'
    elif value in ['Ariel', 'W Motors', 'Shelby', 'TVR', 'Subaru', 'Alpine', 'Ultima']:
        return 'Specialty'
    else:
        return 'Other'

# apply the function to the Car Make column
df['Car Make'] = df['Car Make'].apply(segment_car_make)
```

In [16]:

```
plt.figure(figsize=(10,5))
df['Car Make'].value_counts().plot(kind='bar')
```

Out[16]:

```
<Axes: >
```



In [17]:

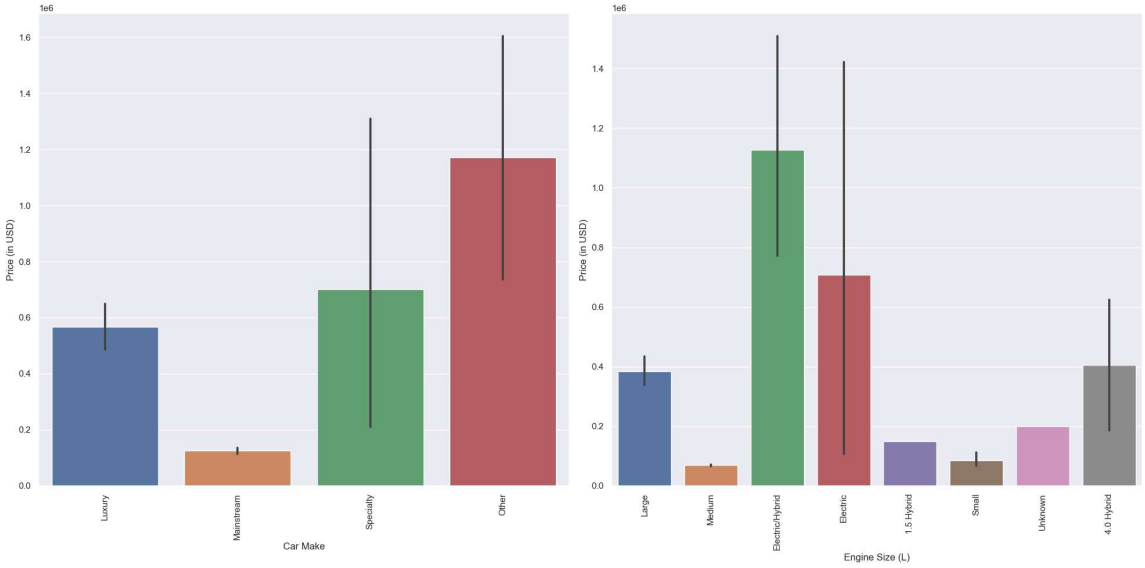
```
df.head()
```

Out[17]:

	Car Make	Year	Engine Size (L)	Horsepower	Torque (lb-ft)	0-60 MPH Time (seconds)	Price (in USD)
0	Luxury	2022	Large	379	331	4.0	101200
1	Luxury	2021	Large	630	443	28.0	274390
2	Luxury	2022	Large	661	561	3.0	333750
3	Mainstream	2022	Large	562	406	32.0	142700
4	Luxury	2021	Large	710	568	27.0	298000

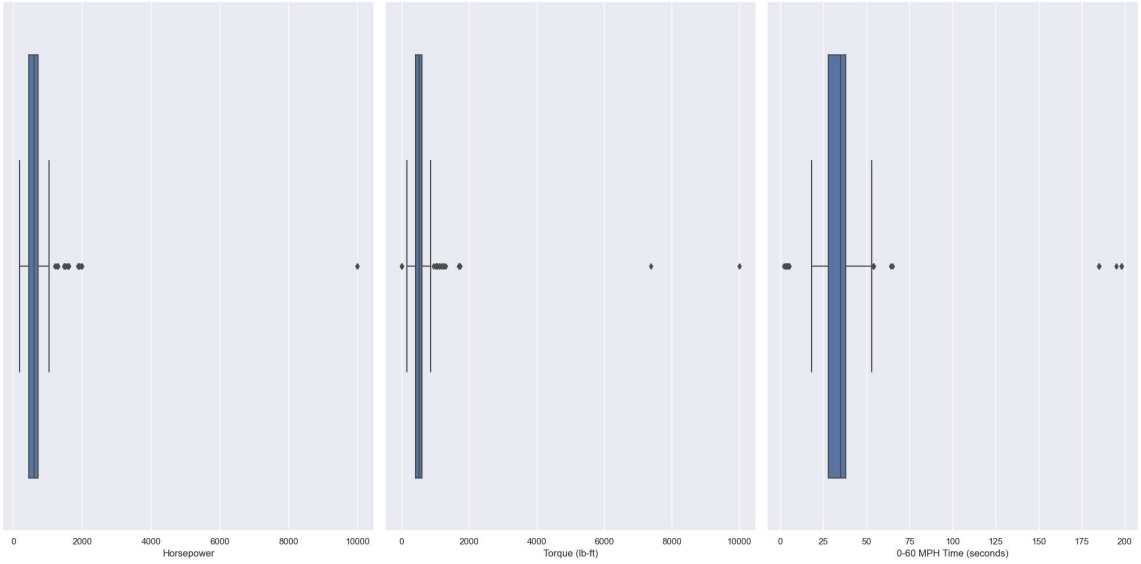
In [18]:

```
# list of categorical variables to plot
cat_vars = ['Car Make', 'Engine Size (L)']

# create figure with subplots
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(20, 10))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.barplot(x=var, y='Price (in USD)', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



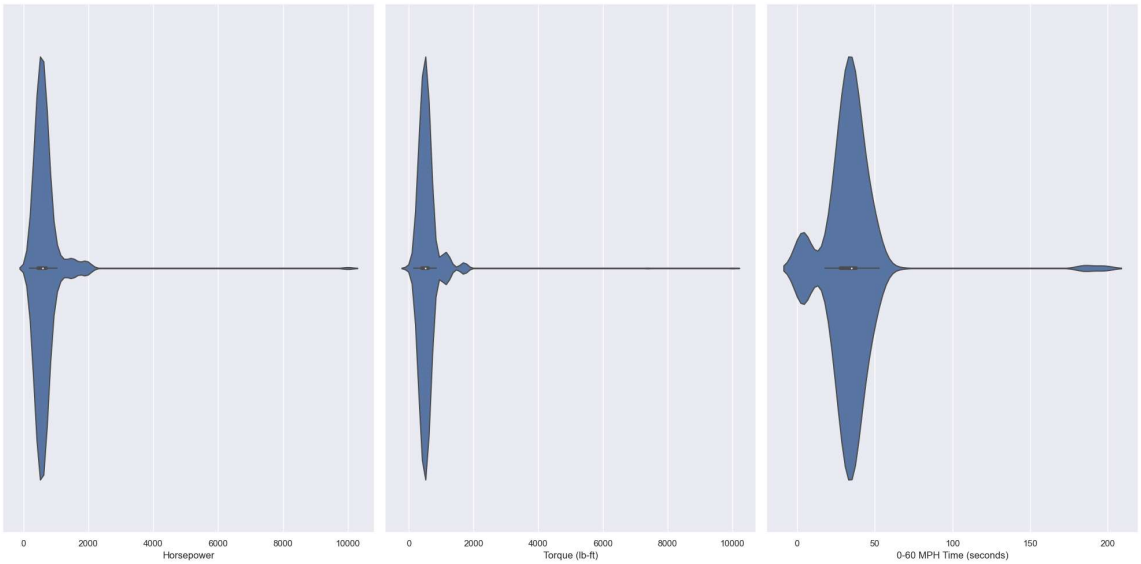
In [19]:

```
num_vars = ['Horsepower', 'Torque (lb-ft)', '0-60 MPH Time (seconds)']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

#Show the boxplot
for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

# adjust spacing between subplots
fig.tight_layout()

plt.show()
```



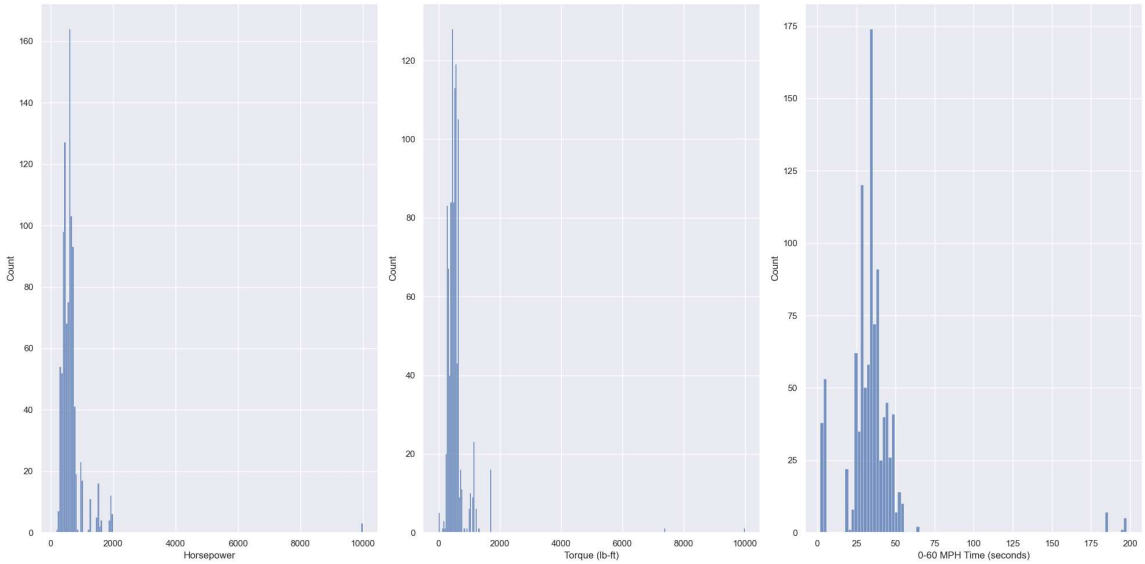
In [20]:

```
num_vars = ['Horsepower', 'Torque (lb-ft)', '0-60 MPH Time (seconds)']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



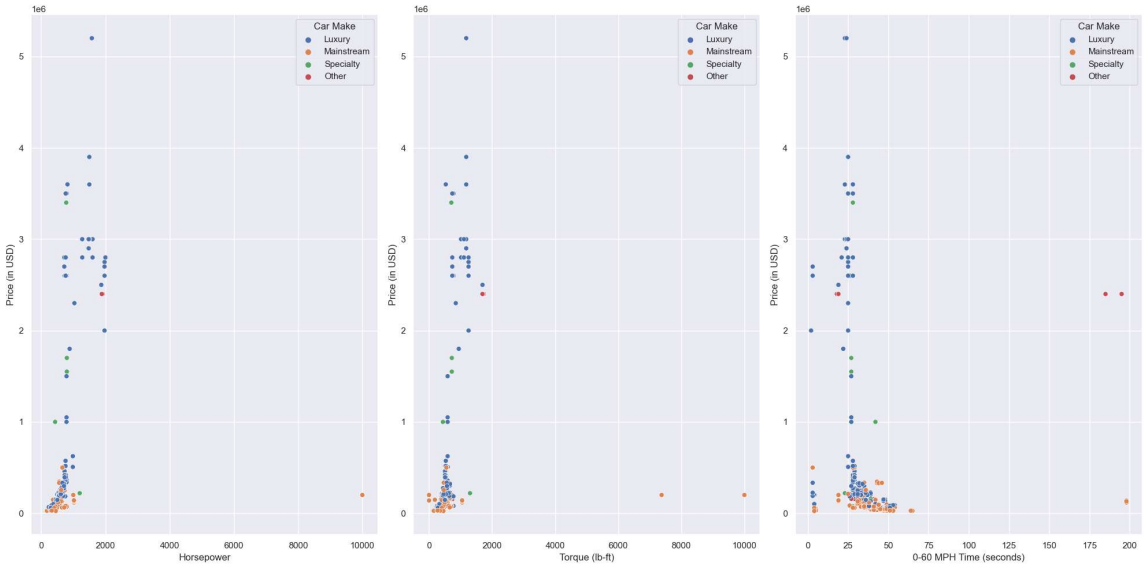
In [21]:

```
num_vars = ['Horsepower', 'Torque (lb-ft)', '0-60 MPH Time (seconds)']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.histplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [22]:

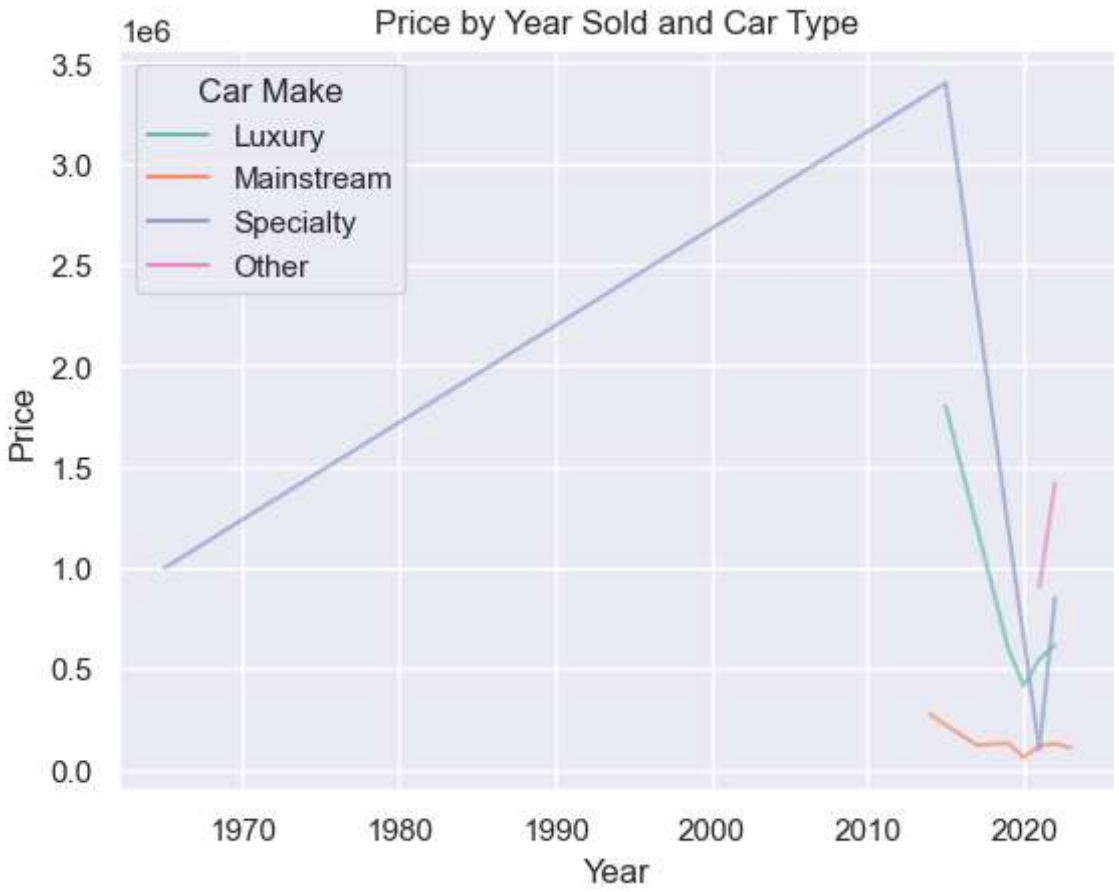
```
num_vars = ['Horsepower', 'Torque (lb-ft)', '0-60 MPH Time (seconds)']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.scatterplot(x=var, y='Price (in USD)', hue='Car Make', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [23]:

```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Year', y='Price (in USD)', hue='Car Make', data=df, ci=None, estimator='

plt.title("Price by Year Sold and Car Type")
plt.xlabel("Year")
plt.ylabel("Price")

plt.show()
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_19324\2942260920.py:4: FutureWa
rning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(x='Year', y='Price (in USD)', hue='Car Make', data=df, ci=N
one, estimator='mean', alpha=0.7)
```



In [24]:

```
df.head()
```

Out[24]:

	Car Make	Year	Engine Size (L)	Horsepower	Torque (lb-ft)	0-60 MPH Time (seconds)	Price (in USD)
0	Luxury	2022	Large	379	331	4.0	101200
1	Luxury	2021	Large	630	443	28.0	274390
2	Luxury	2022	Large	661	561	3.0	333750
3	Mainstream	2022	Large	562	406	32.0	142700
4	Luxury	2021	Large	710	568	27.0	298000

In [25]:

```
#Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[25]:

```
Series([], dtype: float64)
```

In [26]:

```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
Car Make: ['Luxury' 'Mainstream' 'Specialty' 'Other']
Engine Size (L): ['Large' 'Medium' 'Electric/Hybrid' 'Electric' '1.5 Hybri
d' 'Small'
 'Unknown' '4.0 Hybrid']
```

In [27]:

```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
Car Make: [0 1 3 2]
Engine Size (L): [4 5 3 2 0 6 7 1]
```

In [28]:

```
from sklearn.model_selection import train_test_split

X = df.drop('Price (in USD)', axis=1)
y = df['Price (in USD)']

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [29]:

```
# calculate the interquartile range for each feature
q1 = np.percentile(X_train, 25, axis=0)
q3 = np.percentile(X_train, 75, axis=0)
iqr = q3 - q1

# identify outliers using the IQR method
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr
outliers = np.logical_or(X_train < lower_bound, X_train > upper_bound)

# remove the outliers from the training set
X_train = X_train[~np.any(outliers, axis=1)]
y_train = y_train[~np.any(outliers, axis=1)]
```

In [30]:

```
X_train.shape
```

Out[30]:

```
(560, 6)
```

In [31]:

```
y_train.shape
```

Out[31]:

```
(560,)
```

In [32]:

```
X_train.head()
```

Out[32]:

	Car Make	Year	Engine Size (L)	Horsepower	Torque (lb-ft)	0-60 MPH Time (seconds)
767	1	2022	4	444	406	38.0
764	0	2021	4	612	561	34.0
529	1	2021	4	720	590	31.0
252	0	2021	4	592	457	28.0
451	0	2021	4	626	664	33.0

In [33]:

```
y_train.head()
```

Out[33]:

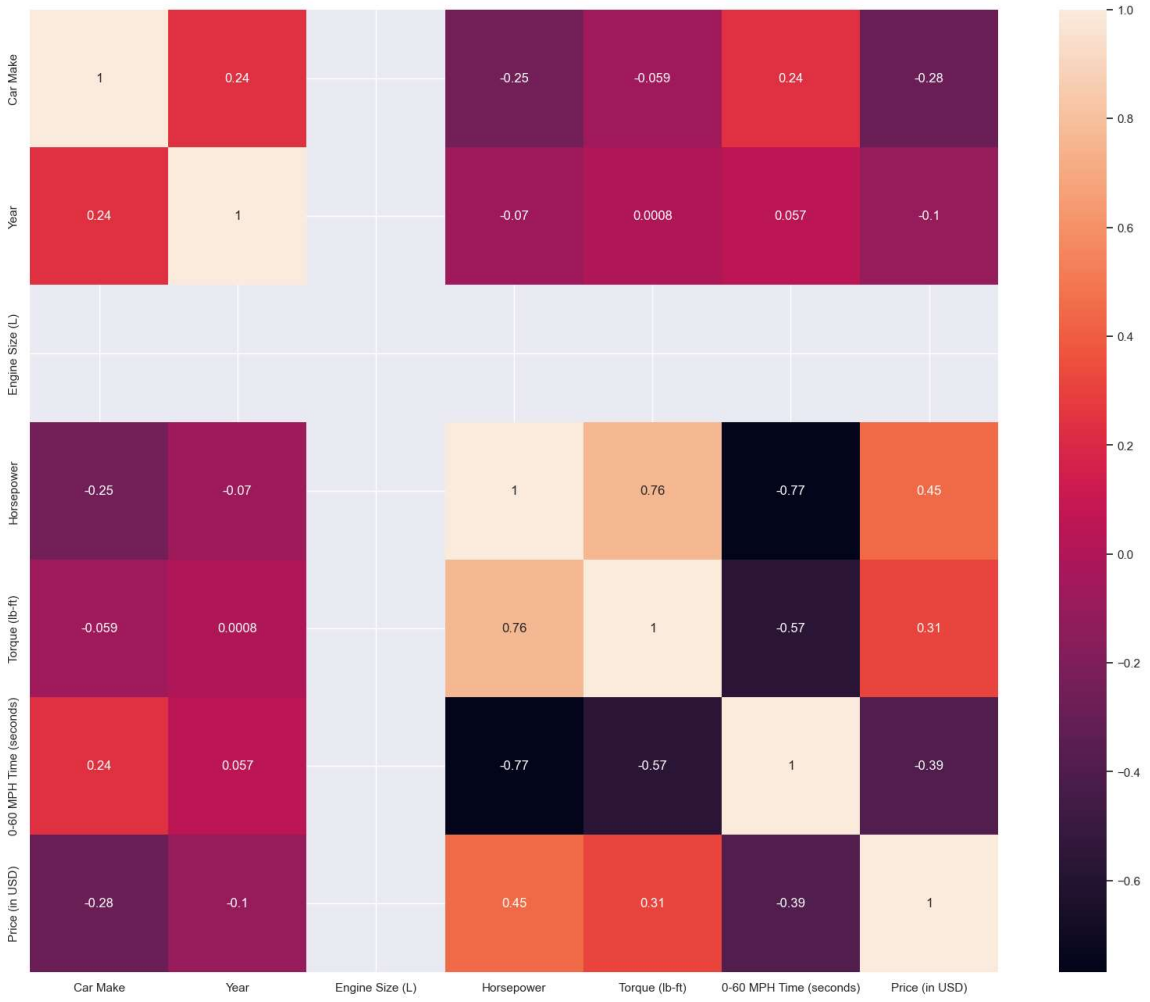
```
767     84595
764    235000
529    325000
252    256500
451    220000
Name: Price (in USD), dtype: int32
```

In [34]:

```
# concatenate X_train and y_train
train_data = pd.concat([X_train, y_train], axis=1)

#Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(train_data.corr(), fmt='.2g', annot=True)
```

Out[34]:

<Axes: >



In [35]:

```
# Remove Engine Size (l) because it has 0 correlation
X_train.drop(columns='Engine Size (L)', inplace=True)
X_train.head()
```

Out[35]:

	Car Make	Year	Horsepower	Torque (lb-ft)	0-60 MPH Time (seconds)
767	1	2022	444	406	38.0
764	0	2021	612	561	34.0
529	1	2021	720	590	31.0
252	0	2021	592	457	28.0
451	0	2021	626	664	33.0

In [36]:

```
# Remove Engine Size (l) because it has 0 correlation
X_test.drop(columns='Engine Size (L)', inplace=True)
X_test.head()
```

Out[36]:

	Car Make	Year	Horsepower	Torque (lb-ft)	0-60 MPH Time (seconds)
799	0	2021	503	505	35.0
311	1	2021	650	650	35.0
85	0	2022	1500	1180	24.0
435	0	2021	1262	1106	25.0
204	0	2022	325	332	53.0

In [37]:

```
X_train.shape
```

Out[37]:

```
(560, 5)
```

In [38]:

```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=8, max_features='auto', min_samp
dtree.fit(X_train, y_train)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\sklearn\tree\_classes.py:277: Fu
tureWarning: `max_features='auto'` has been deprecated in 1.1 and will be
removed in 1.3. To keep the past behaviour, explicitly set `max_features=
1.0'`.
  warnings.warn(
```

Out[38]:

```
DecisionTreeRegressor(max_depth=8, max_features='auto', min_samples_split
=6,
                      random_state=0)
```

In [ ]: